JAVA – Quick Reference

A First Simple Program

```
                                                   comment
/*
   This is a simple Java program.
   Call this file "Example.java".
*/                                          a new class is being defined
class Example {
                                            single-line comment
  // Your program begins with a call to main().
access modifier                       Java applications begin execution by calling main( )
        does not return a value
  public static void main(String args[]) {
                    The keyword static allows main( ) to be called without having to
                    instantiate a particular instance of the class
    System.out.println("Simple Java program.");
  }
}                   outputs the string
```

Java Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Box Class Example

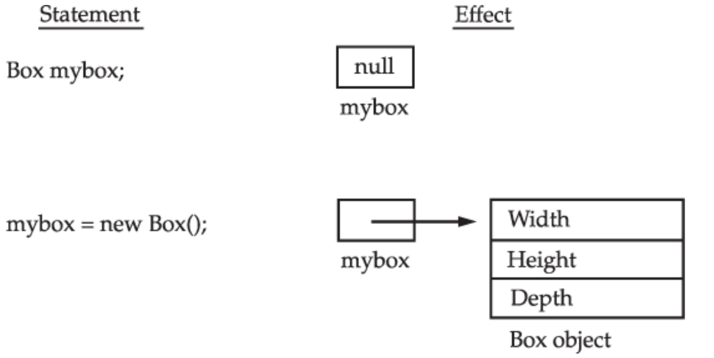
```
/* A program that uses the Box class.
Call this file BoxDemo.java
*/
class Box {
  double width;                       instance variables
  double height;
  double depth;
}

// This class declares an object of type Box.
class BoxDemo {
  public static void main(String args[]) {
    Box mybox = new Box();
    double vol;                       create a Box object called mybox

    // assign values to mybox's instance variables
    mybox.width = 10;   assigns the width variable of mybox the value 100
    mybox.height = 20;  assigns the height variable of mybox the value 20
    mybox.depth = 15;   assigns the depth variable of mybox the value 15
       dot operator links the name of the object with the name of an instance variable
    // compute volume of box
    vol=mybox.width * mybox.height * mybox.depth;

    System.out.println("Volume is " + vol);
  }
}
```

Declaring an object of type Box



Adding a Method to the Box Class

```
// This program uses a parameterized method.

class Box {
  double width;
  double height;
  double depth;

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
                  setDim( ) method is used to set the dimensions of each box
  // sets dimensions of box
  void setDim(double w, double h, double d) {
    width = w;
    height = h;  values of w, h, and d are then assigned to width, height, and depth
    depth = d;
  }
}

class BoxDemo5 {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    Box mybox2 = new Box();
    double vol;

    // initialize each box
    mybox1.setDim(10, 20, 15);
    mybox2.setDim(3, 6, 9);
          10 is copied into parameter w, 20 is copied into h, and 15 is copied into d
    // get volume of first box
    vol = mybox1.volume();
          the value of mybox1.volume( ) is 3,000 and this value then is stored in vol.
    System.out.println("Volume is " + vol);
    // get volume of second box
    vol = mybox2.volume();
    System.out.println("Volume is " + vol);
  }
}
```

Constructors

```
/* Here, Box uses a parameterized constructor to
initialize the dimensions of a box.
*/
class Box {
  double width;
  double height;
  double depth;
have no return type, not even void       Constructors has the same name as the class
  // This is the constructor for Box.
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }
a parameterized constructor that sets the dimensions of a box as specified by those parameters
  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}

class BoxDemo7 {
  public static void main(String args[]) {
  // declare, allocate, and initialize Box objects
    Box mybox1 = new Box(10, 20, 15);
    Box mybox2 = new Box(3, 6, 9);
    double vol;                  values 10, 20, and 15 are passed to the Box( )
                                 constructor when new creates the object
    // get volume of first box
    vol = mybox1.volume();
    System.out.println("Volume is " + vol);

    // get volume of second box
    vol = mybox2.volume();
    System.out.println("Volume is " + vol);
  }
}
```

Overloading Methods

```
// Demonstrate method overloading.
class OverloadDemo {
  void test() {            takes no parameters
    System.out.println("No parameters");
  }

  // Overload test for one integer parameter.
  void test(int a) {          takes one integer parameter
    System.out.println("a: " + a);
  }                two or more methods within the same class that share the same name,
                        their parameter declarations are different
  // Overload test for two integer parameters.
  void test(int a, int b) {  takes two integer parameters
    System.out.println("a and b: " + a + " " + b);
  }

  // overload test for a double parameter
  double test(double a) {    takes one double parameter
    System.out.println("double a: " + a);
    return a*a;
  }           When an overloaded method is called, Java looks for a match between
}             the arguments used to call the method and the method's parameters.

class Overload {
  public static void main(String args[]) {
    OverloadDemo ob = new OverloadDemo();
    double result;

    // call all versions of test()
    ob.test();
    ob.test(10);
    ob.test(10, 20);
    result = ob.test(123.25);
    System.out.println("ob.test(123.25):"+result);
  }
}
```

Overloading Constructors

```
/* Here, Box defines three constructors to
initialize the dimensions of a box various ways.
*/
class Box {
  double width;
  double height;
  double depth;

  // constructor used when all dim. specified
  Box(double w, double h, double d) {
    width = w;
    height = h;           Box takes three double parameters
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {                 Box takes no parameters
    width = -1; // use -1 to indicate
    height = -1; // an uninitialized
    depth = -1; // box
  }

  // constructor used when cube is created
  Box(double len) {          Box takes one double parameter
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}

class OverloadCons {
  public static void main(String args[]) {
    // create boxes using the various constructors
    Box mybox1 = new Box(10, 20, 15);
    Box mybox2 = new Box();
    Box mycube = new Box(7);

    double vol;

    // get volume of first box
    vol = mybox1.volume();
    System.out.println("Vol. of mybox1 is "+vol);

    // get volume of second box
    vol = mybox2.volume();
    System.out.println("Vol. of mybox2 is "+vol);

    // get volume of cube
    vol = mycube.volume();
    System.out.println("Vol. of mycube is "+vol);
  }
}
```

Inheritance Basics

```
// A simple example of inheritance.
// Create a superclass.    a class that is inherited is called a superclass
class A {
  int i, j;

  void showij() {
    System.out.println("i and j: " + i + " " + j);
  }
}
to inherit a class, incorporate the definition of one class into another by using the extends keyword
// Create a subclass by extending class A.
class B extends A {
  int k;                  the class that does the inheriting is called a subclass

  void showk() {
    System.out.println("k: " + k);
  }                       It inherits all of the members defined by the superclass and
                          adds its own, unique elements
  void sum() {
    System.out.println("i+j+k: " + (i+j+k));
  }
}

class SimpleInheritance {
  public static void main(String args[]) {
    A superOb = new A();
    B subOb = new B();

    // The superclass may be used by itself.
    superOb.i = 10;
    superOb.j = 20;
    System.out.println("Contents of superOb: ");
    superOb.showij();
    System.out.println();

    /* The subclass has access to all public
    members of its superclass. */
    subOb.i = 7;
    subOb.j = 8;
    subOb.k = 9;
    System.out.println("Contents of subOb: ");
    subOb.showij();
    subOb.showk();
    System.out.println();

    System.out.println("Sum of i,j, k in subOb:");
    subOb.sum();
  }
}
```

A More Practical Example

```
// This program uses inheritance to extend Box.
class Box {
  double width;
  double height;
  double depth;
  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
  }
  // constructor used when all dim. specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {
    width = -1; // use -1 to indicate
    height = -1; // an uninitialized
    depth = -1; // box
  }

  // constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

BoxWeight inherits all of the characteristics of Box and adds to them the weight component

```
// Here, Box is extended to include weight.
class BoxWeight extends Box {
  double weight; // weight of box
```

extended to include a fourth component called weight

```
  // constructor for BoxWeight
  BoxWeight(double w,double h,double d,double m) {
    width = w;
    height = h;
    depth = d;
    weight = m;
  }
}

class DemoBoxWeight {
  public static void main(String args[]) {
    BoxWeight mybox1=new BoxWeight(10,20,15,34.3);
    BoxWeight mybox2 = new BoxWeight(2,3,4,0.076);
    double vol;

    vol = mybox1.volume();
    System.out.println("Vol. of mybox1 is"+ vol);
    System.out.println("Weight of mybox1 is " +
     mybox1.weight);
    System.out.println();

    vol = mybox2.volume();
    System.out.println("Vol. of mybox2 is "+vol);
    System.out.println("Weight of mybox2 is " +
     mybox2.weight);
  }
}
```

When Constructors Are Called

```
// Demonstrate when constructors are called.
// Create a super class.
class A {
  A() {
    System.out.println("Inside A's constructor.");
  }
}

// Create a subclass by extending class A.
class B extends A {
  B() {
    System.out.println("Inside B's constructor.");
  }
}

// Create another subclass by extending B.
class C extends B {
  C() {
    System.out.println("Inside C's constructor.");
  }
}
```

```
class CallingCons {
  public static void main(String args[]) {
    C c = new C();
  }
}
```

in a class hierarchy, constructors are called in order of derivation, from superclass to subclass.

Method Overriding

```
// Method overriding.
class A {
  int i, j;
  A(int a, int b) {
    i = a;
    j = b;
  }

  // display i and j
  void show() {
    System.out.println("i and j: " + i + " " + j);
  }
}

class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }
```

In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass

```
  // display k – this overrides show() in A
  void show() {
    System.out.println("k: " + k);
  }
}

class Override {
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);
    subOb.show(); // this calls show() in B
  }
}
```

When an overridden method is called from within its subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the superclass will be hidden.

Dynamic Method Dispatch

```
// Dynamic Method Dispatch
class A {
  void callme() {
    System.out.println("Inside As callme method");
  }
}
class B extends A {
  // override callme()
  void callme() {
    System.out.println("Inside Bs callme method");
  }
}
class C extends A {
  // override callme()
  void callme() {
    System.out.println("Inside Cs callme method");
  }
}
class Dispatch {
  public static void main(String args[]) {
    A a = new A(); // object of type A
    B b = new B(); // object of type B
    C c = new C(); // object of type C
    A r; // obtain a reference of type A
    r = a; // r refers to an A object
    r.callme(); // calls A's version of callme
    r = b; // r refers to a B object
    r.callme(); // calls B's version of callme
    r = c; // r refers to a C object
    r.callme(); // calls C's version of callme
  }
}
```

the version of callme( ) executed is determined by the type of object being referred to at the time of the call. Had it been determined by the type of the reference variable, r, you would see three calls to A's callme( ) method.

Using Abstract Classes

to declare a class abstract, you simply use the abstract keyword in front of the class keyword at the beginning of the class declaration

```
// A Simple demonstration of abstract.
abstract class A {
  abstract void callme();
```

abstract classes can include as much implementation as they see fit

```
  // concrete methods are still allowed in
  // abstract classes
  void callmetoo() {
    System.out.println("A concrete method.");
  }
}
class B extends A {
  void callme() {
    System.out.println("B's impl. of callme.");
  }
}

class AbstractDemo {
  public static void main(String args[]) {
    B b = new B();
    b.callme();
    b.callmetoo();
  }
}
```

Interfaces

```
interface Callback {
  void callback(int param);
}
```

the methods that are declared have no bodies

variables can be declared inside of interface declarations, they are implicitly final & static

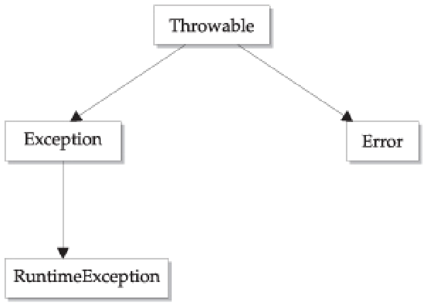
```
class Client implements Callback {
// Implement Callback's interface
public void callback(int p) {
  System.out.println("callback called with " + p);
}
```

the methods that implement an interface must be declared public

```
void nonIfaceMeth() {
  System.out.println("Classes that implement
     interfaces " + "may also define other
     members, too.");
}
}

class TestIface {
  public static void main(String args[]) {
    Callback c = new Client();
    c.callback(42);
  }
}
```

Exception Types



Uncaught Exceptions

```
class Exc0 {
  public static void main(String args[]) {
    int d = 0;
    int a = 42 / d;
  }
}
```

When the Java run-time system detects the attempt to divide by zero, it constructs a new exception object and then throws this exception

any exception that is not caught by your program will ultimately be processed by the default handler

Using try and catch

```
class Exc2 {
  public static void main(String args[]) {
    int d, a;
```

to guard against and handle a run-time error, simply enclose the code that you want to monitor inside a try block

```
    try { // monitor a block of code.
    d = 0;
    a = 42 / d;
```

an exception is thrown, program control transfers out of thetry block into the catch block

```
    System.out.println("This won’t be printed.");
```

include a catch clause that specifies the exception type that you wish to catch

```
    } catch (ArithmeticException e) {
    // catch divide-by-zero error
     System.out.println("Division by zero.");
   }
```

Once the catch statement has executed, program control continues with the next line in the program following the entire try / catch mechanism.

```
   System.out.println("After catch statement.");
  }
}
```

Multiple catch Clauses

```
// Demonstrate multiple catch statements.
class MultipleCatches {
  public static void main(String args[]) {
    try {
      int a = args.length;
      System.out.println("a = " + a);
      int b = 42 / a;
      int c[] = { 1 };
      c[42] = 99;
    } catch(ArithmeticException e) {
      System.out.println("Divide by 0: " + e);
    } catch(ArrayIndexOutOfBoundsException e) {
      System.out.println("Array index oob: " + e);
    }
    System.out.println("After try/catch blocks.");
  }
}
```

When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed.

after one catch statement executes, the others are bypassed, and execution continues after the try / catch block.

Creating a Thread

```
class NewThread implements Runnable {
  Thread t;
  NewThread() {
```

an instance of a class that implements the Runnable interface

```
    // Create a new, second thread
    t = new Thread(this, "Demo Thread");
    System.out.println("Child thread: " + t);
    t.start(); // Start the thread
  }
```

start( ) executes a call to run( )

```
  public void run() {
  try {
```

establishes the entry point for another, concurrent thread

```
    for(int i = 5; i > 0; i--) {
      System.out.println("Child Thread: " + i);
      Thread.sleep(500);
    }
```

causes the thread to suspend execution for the specified period of milliseconds

```
    } catch (InterruptedException e) {
      System.out.println("Child interrupted.");
    }
    System.out.println("Exiting child thread.");
  }
}
class ThreadDemo {
  public static void main(String args[]) {
  new NewThread(); // create a new thread
  try {
    for(int i = 5; i > 0; i--) {
      System.out.println("Main Thread: " + i);
      Thread.sleep(1000);
    }
```

causes the thread to suspend execution for the specified period of milliseconds

```
    } catch (InterruptedException e) {
      System.out.println("Main thread int.");
    }
    System.out.println("Main thread exiting.");
  }
}
```